BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

Safe and Protected Execution for the Morph/AMRM Reconfigurable Processor

Andrew A. Chien
Department of Computer Science and Engineering
University of California, San Diego
achien@cs.ucsd.edu

Jay H. Byun
Department of Computer Science
University of Illinois at Urbana-Champaign
jaybyun@cs.uiuc.edu

April 1, 1999

Abstract

Technology scaling of CMOS processes brings relatively faster transistors (gates) and slower interconnects (wires), making viable the addition of reconfigurability to increase performance. In the Morph/AMRM system, we are exploring the addition of reconfigurable logic, deeply integrated with the processor core, employing the reconfigurability to manage the cache, datapath, and pipeline resources more effectively. However, integration of reconfigurable logic introduces significant protection and safety challenges for multiprocess execution. We analyze the protection structures in a state of the art microprocessor core (R10000), identifying the few critical logic blocks and demonstrating that the majority of the logic in the processor core can be safely reconfigured. Subsequently, we propose a protection architecture for the Morph/AMRM reconfigurable processor which enable nearly the full range of power of reconfigurability in the processor core while requiring only a small number of fixed logic features which to ensure safe, protected multiprocess execution.

1. Introduction

Trends in semiconductor technology suggest that the use of reconfigurable logic blocks within the processor will be desirable in the future. Projections from Semiconductor Industry Association(SIA) for the year 2007 indicate advanced semiconductor processes using 0.1 micron feature sizes [1]. However, this feature size, as measured by transistor channel length, is of decreasing importance to logic and circuit as well as processor speed. In systems of that era, logic density, logic speed, and processor speed will be dominated by interconnect performance and wiring density. For 2007, the SIA projects pitch for the finest interconnect at 0.4-0.6 microns. Between logic blocks, average interconnect lengths typically range from from 1,000x to 10,000x pitch – up to 6mm of intra-chip interconnect length. For such an interconnect, the achievable global clock speed would be limited to approximately 1 nanosecond. Within a few technology generations, a crossover will occur, and the average interconnect delay will surpass logic block delays -- projections indicate that by the year 2007, average interconnect delay can be equivalent to five gate delays. Once past the cross-over point, dynamic interconnect (reconfigurable interconnect or logic) can be introduced at modest impact even on critical timing paths[2]. In such systems, the dynamic configurability in the processor can be used to significant advantage [4, 5], improving performance by factors of 10 to 100x for computational kernels while avoiding the traditional disadvantages of custom computing approaches such as I/O coprocessor coupling and slower logic [6]. In these systems, reprogrammable logic blocks will replace static interconnects in the processor core, paving the way for a new class of architectures which are customized to the application, delivering more robust and higher performance.

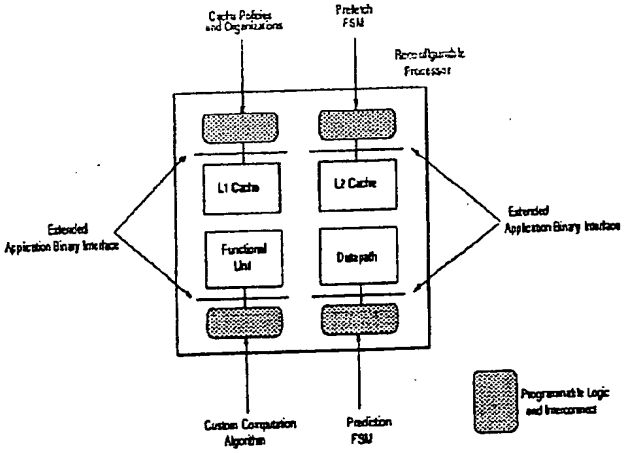

Figure 1. Reconfigurability in the processor core and the extended application to fixed hardware interface

Reconfigurable, or application adaptive processors allow customization of mechanisms, bindings, and policies on a per application basis. While current microprocessors implement a number of aggressive architectural techniques such as speculative execution, branch prediction, block prefetching, multi-level caching, etc. to achieve higher execution speeds, these mechanisms and policies are tuned

0-7695-0375-6/99 $10.00 © 1999 IEEE

for a broad suite of applications (e.g. SPEC), and thus cannot be tightly matched to the needs of a particular application, procedure, or even loop in an application. For example, the cache block size and organization is chosen to maximize performance over a suite of applications, but may not give best performance on any particular application. Similar constraints apply to other performance critical aspects such as value prediction, branch prediction, and data movement. In contrast, a processor incorporating reconfigurability can adopt optimal policies (and in some cases better mechanisms) for the application, enabling increased execution efficiency. Thus, the reconfigurable logic can used to tune the processor to better match the application, rather than the more traditional view of thinking of it as an add-on coprocessor. One example of this per-application basis tuning would be to adapt the cache line size to maximize performance for that application[3]. This approach is embodied in the Morph/AMRM (Adaptive Memory Reconfiguration Management) architecture [4, 5], and the basic change in perspective is that the reconfigurable hardware is an extension of the application program, extending the application – fixed hardware interface to enable more efficient execution. The fixed hardware then has a somewhat richer (and in parts lower level) interface as shown in Figure 1. Studies of Morph/AMRM have demonstrated that performance increases of ten to 100 times are possible [5]. In essence, this is an extension of the application binary interface (so-called ABI), but need not be a non-portable extension of the application programming interface (API) if appropriate CAD support is available. This approach is similar to that which has recently gained popularity in the software design community as "open implementations" [7] in which software architects recognize the need to open the implementation for customization for particular application uses in order to achieve adequate performance.

Introducing application-controlled reconfigurability in the processor raises significant challenges for ensuring process isolation and protection (multiprocess isolation), a critical element of robust desktop and to an increasing degree, embedded computing systems. Multiprocess isolation is an essential modularity element in software systems: without the guarantee of safely isolated and protected processes, the system can never be robust since software faults cannot be contained and the system cannot be safely extended. It is essential for robust reconfigurable computing that an application's customization only affect its computation, not that of other applications. For example, if application-defined hardware were allowed to control hardware addressing, it could allow unauthorized corruption of operating system data or even the data of other application processes. If an application-defined hardware were allowed to control data prefetching, it could swamp the memory system with spurious requests. If application-defined hardware were allowed to control privilege mode changes, it could compromise all traditional protection structures.

Our study examines the protection structures of traditional processors and operating systems, and based on these lessons, proposes a safe multiprocess execution architecture for reconfigurable systems. We analyze in detail the software and hardware mechanisms central to the process protection in conventional processors and OS, specifically studying the MIPS R10000 [8] microprocessor, an exemplar of a system employing Unix/RISC protection architecture. This study elucidates the key mechanisms and architectural features for Unix style two mode protection, and addressing based isolation. The key feature of this protection architecture is process isolation via address isolation and mediation. Specifically,

1. All access to hardware devices is mediated by the operating system,
2. The operating system manages address translation to isolate processes,
3. Application processes cannot change the address translation information,
4. Application processes cannot substitute other translation information,
5. All application accesses are subject to this translation, and
6. The hardware ensures these guarantees

We subsequently describe the Morph/AMRM architecture, outlining the dimensions of configurability and the hazards for multiprocess protection they induce. For the Morph/AMRM system, we then describe the protection architecture, describing in detail how each of the key properties of the operating system / processor protection architecture are provided. The key elements of this protection architecture are:

1. A hardwired control processor which controls instruction sequence and privilege mode transitions
2. A hardwired control processor to TLB control for address translation and TLB entry management
3. A requirement for all other configurable elements (system chip sets, input/output devices, memory controllers) must deal in virtual addresses, and their accesses are checked by local TLBs
4. Controlled access to key shared interconnects such as the system bus are controlled by hardwired arbiters which are not changed, system reserves highest priority to allow preemption for these resources

This architecture enables configurability in the processor complex because it can ensure multiprocess protection (safe configuration). We also believe it enables much of the useful configurability in the processor complex, notably policies for improving efficient management of resources and even the addition of instructions, special functional

units, or even processor state. The model provided to application programs is a private, configurable, virtual machine which enables rich application customization. These applications (and their customizations) are cleanly isolated.

The remainder of the paper is organized as follows. Section 2 describes the basic problem of protected execution and process isolation in computer systems. Section 3 describes our analysis of the software and hardware mechanisms central to the process protection in conventional processors and operating systems. Section 4 discusses the implications of reconfigurability on process protection and identifies the key requirement for safe process isolation in reconfigurable processors. In Section 5, we describe the Morph/AMRM system and a proposed protection architecture that meets these requirements set forth in Section 3. Section 6 discusses alternate approaches and the limitations on configurability imposed by the Morph/AMRM protection architecture. Sections 7 summarizes future work and the material covered in this paper.

2. Process Isolation: the Problem

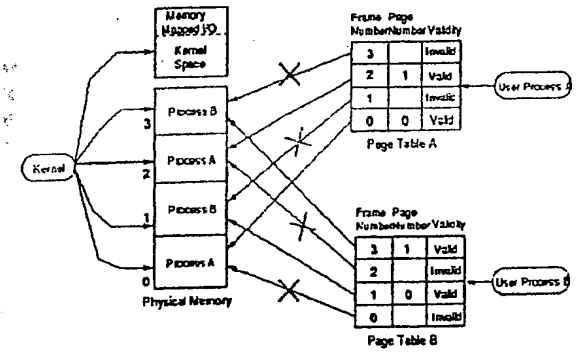

Figure 2: Multiprocess Protection based on Address Space Isolation

To understand the challenges of multiprocess isolation, it is instructive to first consider the possible modalities in which multiprocess isolation can be compromised. In the simplest mode, an application corrupts the data of another, causing it to fail or compute incorrectly. In a more complex mode, the application somehow locks up the machine, so no other application state is damaged, but neither can the machine make progress. One example of this would be jamming the memory bus or defeating the timer interrupt which ensures preemption. A more serious failure mode is to corrupt the operating system's data, which can lead to a machine crash in which all applications have data corruption. Finally, an application could also corrupt input/output device state, confounding the operating system, the device (leading to data loss or misdirection), or application data itself. In all of these cases, the failure is the result of allowing an application action which can affect the machine hardware state, other application memory state, or operating system state.

The key issue in safe multiprocess execution is to control access to hardware resources, ensuring that these accesses are non-interfering. In general, access to main memory, as well as other architecturally visible state (processor data registers, control registers), system chip registers, and input/output device state must be controlled. Traditional approaches partition memory access, virtualize resources such as processor data resources with multitasking, and use operating system calls to mediate operations which require access to control registers, system chip sets, input/output device state, etc. Note that isolation and virtualization must apply to any resource at any level that a process can claim its ownership. The final piece of the puzzle is that in order to support the virtualization and multitasking, transitions between the different entities must be carefully controlled to prevent compromise.

3. Process Isolation in the MIPS R10000

The key issue in maintaining a safe multiprocessing environment is ensuring process isolation: the processor and the OS must prevent independent processes from interfering with the data and memory of each other and of the operating system kernel. They must also prevent a malicious process from taking over the processor and locking up the system.

Through a detailed analysis of the R10000 architecture and operating system, we identify the hardware mechanisms and OS software structures that are central to process isolation. We chose the MIPS R10000 processor as an exemplar of a modern RISC processor that supports a relatively simple UNIX style protection structure [9]. We first examine how a UNIX style operating system ensures process isolation and thereby derive the hardware requirements it imposes. Then identify the corresponding support in the R10000 processor. In the following discussion, we assume that the address translation is on a simple paging system. Most of today's systems actually employ multiple-paging or segmented paging but the address translation mechanism is fundamentally the same as a simple paging system.

3.1 Operating System-based Process Protection

3.1.1 Application and Operating System Memory Isolation

Application and operating system memory isolation is achieved through controlled address translation. The physical memory of each process is isolated by having process's virtual address space pages map to its own physical memory frames only. To protect processes from modification by other processes, the memory-management hardware and the OS must prevent programs from changing their own address mappings. The UNIX kernel, for example, runs in a privileged mode (kernel mode or system mode) in which memory mapping may be controlled, whereas application processes run in an unprivileged mode (user mode). The page tables, mapping information for each process reside in the memory space of the kernel so that they can only be modified by the OS running in kernel mode This address translation control to ensure isolation is achieved through the following mechanisms in UNIX [9, 10].

1. Locating correct translation information for each process.
   By using a special page table base register(ptbr) which is set from the process control block(PCB) on each process switch, the OS can correctly locate the page table for the executing process. Then the index portion of the virtual address is added to the address pointed to by the ptbr to locate the appropriate page table entry (PTE).
2. Distinguishing valid and invalid entries in page tables.
   Notice that the page table can contain entries that are not used by the process. These unused entries correspond to the pages that are not in the process's logical address space and thus compromise process isolation. The OS uses *valid-invalid* bits to distinguish these entries. Alternatively, the page table can also be implemented to contain only the entries that are actually used by the process. This implementation will require a special register containing the length of the process's page table; usually called *page-table length register(PTLR)*. PTLR can be used to check if page index portion of the virtual address is in the range and therefore is not accessing illegal translation information.
3. Controlling access types
   While the address translation to physical memory frames can be valid, the access to those physical memory frames are unlimited; the process can read, write, and execute them. It will be safer and more efficient if we can control the type of access to them. The protection bit field in the PTE provides this access control information. At the same time that the physical address is being computed, the protection bits can be checked to verify that no accesses not granted are being made. These bits usually indicate whether the process can read/write, read-only, or execute-only. The type and the number of the protection bits provided are dependent on the underlying processor.
4. Managing TLB consistency.
   The translation information, namely the PTE, is cached in the processor's TLB to avoid extra memory access to the page table. Using special privileged instructions, OS updates the TLB with consistent mapping information when a miss occurs. But notice that after a context switch, although the new page table is pointed to by the new process's *ptbr*, the TLB would contain entries that are left over from the previous processes. Therefore, to ensure process isolation, we need to invalidate or distinguish the entries in the TLB that does not belong to the executing process. This can be done by allowing the OS to flush the TLB by a special privileged instruction after a context switch or by tagging the TLB entries with the process ID's and valid-invalid bits.

3.1.2 Resource Protection through Operating System Mediation

Not only the memory but also all resources that can be shared by processes must be isolated and virtualized. The OS provides protected resource access through mediation. The most fundamental role of the operating system is to mediate process's accesses to system resources. Processes are provided with a system call interface to the operating system kernel, and all accesses to the resources must go through the system calls to the kernel hence protecting the resources from illegal accesses of processes. The operating system can enforce this through the following features of the OS and the hardware:

1. System trap instruction and system call handler:
   System call invocation is made through special trap instruction that changes the mode to kernel mode and jumps to system call handler location predefined by the OS. This system call handler is responsible for all system call processing in kernel, such as saving/restoring process context, selecting appropriate kernel function through system call dispatch vector, transferring control back to user process in user mode. The system call handler is one of the most fundamental routines in the kernel and is written very carefully to ensure safety.
2. Interrupt architecture:
   The interrupt architecture in the processor and the OS guarantees correct invocation and handling of interrupts and provides priority management mechanisms. The interrupt handler is one of the most fundamental and carefully-written kernel routines and is responsible for safe mode transition, context saving/restoring, and priority based servicing.

For general I/O resources, the following features of the OS and the hardware ensures protection.

3. Privileged I/O instructions:

I/O address space is separate from main memory space(e.g. x86 processors) and can only be accessed through privilege mode instructions(e.g. inb, outb in x86)

4. Memory mapped I/O in protected memory space:
   I/O accesses are made by memory access instructions to main memory space, but this space can only be accessed by kernel.
5. I/O buffers in protected memory space:
   Buffers for I/O operations reside in kernel space or space private to each process and thus protected from other processes.

For CPU resources,

6.Preemptive time-quota based scheduling:
   A process must relinquish the CPU when its time-quota expires. The scheduler is designed carefully to avoid starvation of low-priority processes. Timer interrupt has a very high priority, second only to power-failure interrupt.

3.1.3 Machine State Virtualization and Safe Transitions (context switch)

Multiprocess isolation in a computer system can be considered as providing to each process a private and isolated virtual machine. The OS captures the state of each virtual machine provided and ensures safe transitions between virtual machine states (i.e. safe context switching). In UNIX, the virtual machine state is captured in the *Process Control Block(PCB)*. It contains a snapshot of general-purpose registers, memory context, and special registers, etc. Context switching involves a series of low-level privileged instructions to switching these states and performing many hardware-specific tasks in order to ensure safe transition to a new virtual hardware state. These hardware-specific tasks include flushing the data, instruction, address translation(TLB) caches, and flushing the execution pipeline.

The OS process isolation mechanisms that are identified in this section can be distilled into two key elements in the hardware which enable process isolation:

1. Processor execution modes and kernel address space
2. Control of address translation and TLB management

3.2 Hardware Support for Process Protection

The two key elements in the hardware to support process protection can be further classified into a range of features that must be provided by the hardware to enable process protection mechanisms dictated by the OS:

Execution modes and kernel address space: The processor should at least provide two modes of execution, i.e. privileged execution mode and user mode, so that the kernel data structure, special registers, and processor control bits can only be access and altered through special privileged instructions. The virtual address space should contain a kernel address space that can only be accessed in privileged execution mode. This is where the kernel data structure resides.

Context register: This register identifies the current process and is used to select appropriate page tables for controlling memory access.

Valid-Invalid bit, PTLR: The processor should be able to recognize the valid-invalid bit for PTEs which identifies those that do not map to a valid physical address. Optionally, process can have a Page Table Length Register to set the bound on the page table.

Protection bits: The process should be able to recognize a protection bit or some set of them to allow a finer level of access control on the pages.

TLB tagging or flush mechanism: The processor should enable the OS to distinguish the TLB entries that belong to the executing process by having TLB entries tagged with process ID's or allow the OS to flush out the TLB by supplying a special TLB-flush instruction.

In this section, we discuss how these features are implemented in the R10000 processor, a typical superscalar RISC microprocessor from the MIPS RISC family.

3.2.1 The System Control Processor (CP0)

The central part of R10000's protection architecture is the CP0 processor. The CP0 controls execution mode switch, TLB management and control, exception catching and dispatching, cache control, and the TLB where the protection checking is carried out. CP0's states and registers can only be altered by CP0 instructions, which are privileged MOVE instructions(*MFC0, MTC0,* etc). These instructions can only be executed in kernel mode. Thus the system is protected from non-privileged processes which attempt to alter the CP0 processor state including the processor operation mode

3.2.2 Processor Modes

Processor operating modes for the R10000 include Kernel mode, Supervisor mode, and User mode. The current mode is indicated in the CP0 registers, and that mode can only be changed in two ways:

1. CP0 status register's *KSU* field is changed explicitly by CP0 MOVE instructions executed in kernel or supervisor mode.
2. The processor is handling an error (*ERL* bit in CP0 is set) or an exception(*EXL* bit in CP0 is set) and is forced into kernel mode. This mechanism is used to implement guarded transitions such as those used by system calls.

The current operating mode also determines access to the kernel address space (or a respective application address space) as described in the next section.

When the system starts up, the system is in Error level(*ERL* bit is set) so the processor is in the kernel mode. The cold-reset exception handler boot straps the operating system which then runs in kernel mode.

3.2.3 Kernel Address Space

To enable the operating system kernel to mediate access to all hardware resources as well as interprocess communication, it must have access to all memory. While in kernel mode, the processor is allowed to access all kernel segments (*kseg0*, *kseg1*, *kseg3*) and user segments, allowing access to all of the system resources.

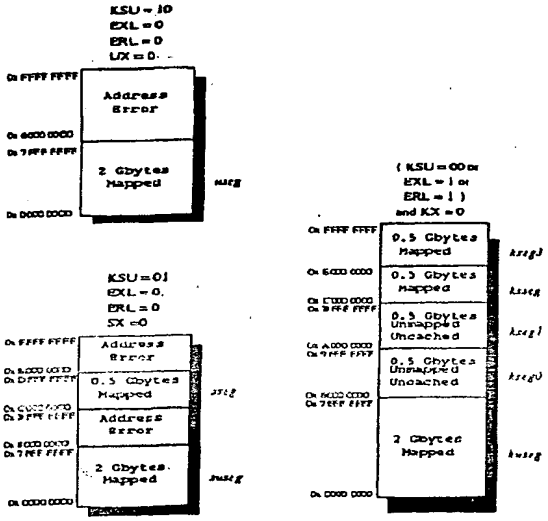

Figure 3. Kernel and User Address Spaces in the R10000

While in user mode, the processor can only access a subset of the memory space as determined by the address mappings for that application process. This is typically a subset of the address space, and is illustrated in Figure 3. The accessible address space for a user process is determined by the TLB entries whose address space identifier (ASID) tags match the ID of the process. If an application process attempts to reference an address not mapped by its TLB entry or attempts to reference an address in kernel address space, an Address Error Exception will occur. As with all exceptions, this is handled by an operating system installed exception handler and generally results in a fatal signal for the application process and its termination, thus protecting the system data and other processes' data from unauthorized application access.

3.2.4 Control of Address Translation and TLB Management

The control of address translation, namely checking validity and access type control, is supported in R10000 by moving the PTE to CP0 registers and then performing corresponding checks and raising appropriate exceptions in the CP0 control processor core. The *EntryHi* and *EntryLo* CP0 registers are always loaded with the TLB entry or the page table entry(if the TLB misses) that corresponds to the virtual address. The address translation, as well as the required checking and exception raising, is done using the contents of these registers. The *EntryHi* and *EntryLo* CP0 registers are loaded only through *TLBP*, *TLBR* CP0 instructions(in case of TLB hit) or generic *move to CP0* instructions(in case of TLB miss) so that these registers cannot be altered by user processes.

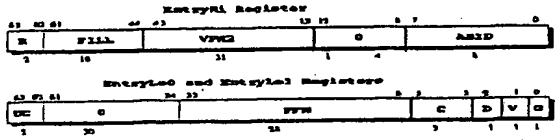

Figure 4. The Address Translation Control registers in the R10000.

As shown in Figures 4, the TLB entry and the correspoding *EntryHi* and *EntryLo* CP0 registers have validity bits V and D that are recognized by the CP0 and used as the invalid-valid bit and the protection bit that were described in the section on the OS protection scheme.

The CP0 has *Context* and *Xcontext* registers which point to the base of the page tables so that the page table for the executing process is located safely and quickly after a context switch.

The R10000 provides to the OS the means to manage and maintain consistent TLB entries. The TLB entries are not flushed after every context switch in R10000. Instead, R10000 allows the TLB entries loaded for different processes to be distinguished. Support for this can be found in the 8-bit *ASID*(Address Space ID) field in the TLB entry. *ASID* is a unique id that is assigned to each process. After a context switch, a new value is loaded into the *ASID* field of the *EntryHi* register. Only the TBL entries whose ASID tag matches this ID or set to global are enabled. In this way, it is guaranteed that only the pages that belong to the executing process or the pages that are globally shared are translated and accessed. The TLB entries are written with the contents of the *EntryHi* and *EntryLo* CP0 registers only through *TLBP*, *TLBWI*, *TLBWR* CP0 instructions so that the user processes cannot alter the TLB directly.

The L1 caches in R10000 are virtually indexed and physically tagged. It is virtually indexed in order to reduce access time to the cache by allowing finding set/reading tag and address translation for tag to occur concurrently. It is

important to note that because it is still physically tagged, accesses to the cache cannot bypass the TLB, where most of memory protection scheme is implemented, even though it is virtually indexed

4. Process Isolation in Reconfigurable Hardware

Because multiprocess decomposition is a critical element of modularity and fault isolation in software systems, providing a safe multiprocess execution is a critical requirement for reconfigurable processors to achieve widespread use. We have described the basic multiprocess protection problem in Section 2, and outlined the possible failure modes. In reconfigurable systems, these failure modes are largely the same, but can occur via the actions of both the software application program and the application-adapted configurable hardware. As we will see, a key aspect of a protection architecture for reconfigurable systems is to restrict the capabilities of the configurable hardware for unchecked access to architecturally visible and invisible system state.

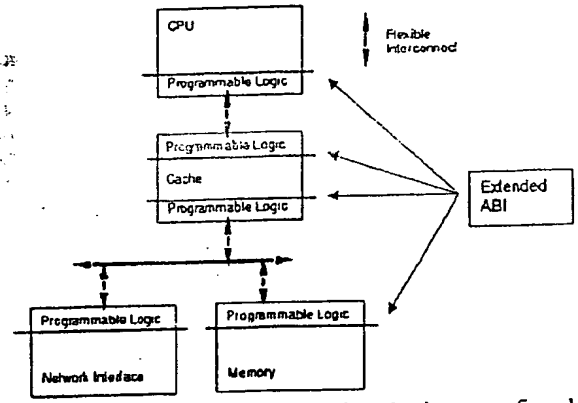

Figure 5: The canonical application-adaptive reconfigurable architecture, where elements of reconfigurable logic can in general be attached to all elements of interconnect, logic, and memory in the system.

4.1 Reconfigurable Architecture Model

We characterize reconfigurable processors as a new class of processors with a fraction of the silicon area dedicated to reconfigurable logic blocks on which application-customized mechanisms or computations can be built. This basic architecture is characterized in Figure 5. In this basic architectural framework, reconfigurable elements can be attached to all elements of interconnect, logic, and memory, enabling any conceivable augmentation of the hardwired system. This is the most general model, and is the starting point for our analysis of process isolation. As examples of the type of configurability that can be achieved, major functional blocks in these processors can also be reconnected, replaced, or have their communication mediated. Elements of state can be altered, arbitration protocols can be changed, finite-state machines can be replaced and interconnect resources can be added (or diverted) to speed (or slow) particular data movement operations. All of these changes can be integrated into the functional operation of the processor (e.g. change the meaning of an instruction) as well as its protection structure (e.g. allow a non-privileged instruction to change a CP0 register or a range of TLB entries). In summary, in the most general case, the configurable hardware can be attached to any part of the entire system, its actions can affect any part of the hardware system.

4.2 Implications of Reconfigurability on Process Protection

For safe protected execution in reconfigurable machines, we need the guarantee of process isolation that the rigid process isolation mechanisms provide while allowing a certain degree of freedom in reconfigurability of the hardware. Reconfigurability adds to the conventional concerns of controlling the software <-> software interactions of processes that share the processor, resulting in the following range of concerns:

1. Software <-> software interactions
2. Software <-> configurable hardware interactions
3. Configurable hardware <-> configurable hardware interactions

These cases are illustrated in Figure 6. The first case corresponds to the traditional process protection problem. In the second case, as the processor is context switched amongst the application processes, the surrounding configurable hardware may or may not be switched synchronously. In fact in some cases, it may be clearly advantageous for the configurable hardware to continue execution while the corresponding application process is context switched out. Because the configurable hardware is properly viewed as an extension of the application process' "virtual machine", care must be taken to ensure that inappropriate interactions do not occur. For example, one such reconfiguration might involve permuting data in the memory between phases of execution in an application program. While it might be advantageous to allow this permutation to go on while the application is not scheduled on the processor, process isolation dictates that the customization of the memory controller must not affect the functional behavior of the system for other processes (e.g. other applications or even the kernel). Finally, the third

case involves interactions of the configurable hardware with shared system (hardwired) resources which cause either compromises of data or more basic aspects of the system. For example, if application #1 reconfigures the addressing interface from the processor to the memory bus, and application #2 customizes the addressing interface of the memory controller, allowing direct interaction could cause inappropriate data access or corruption. In short, the reconfigured logic as well as the processes must be safely isolated to achieve a robust and extensible system.

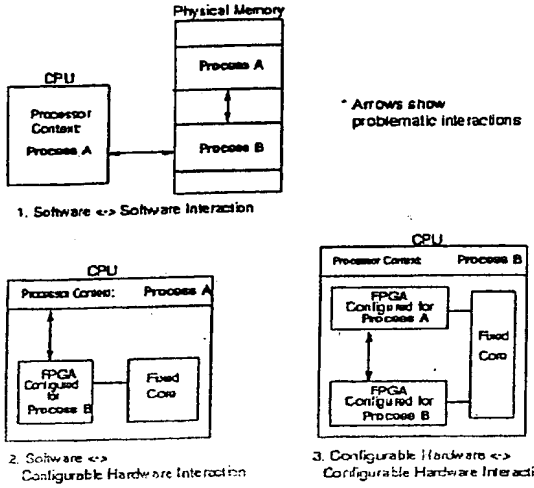

Figure 6. Three types of interactions can cause protection compromises in application-adaptive configurable machines. Arrows show problematic interactions.

Requirements for process isolation in a configurable architecture extend the hardwired system requirements outlined in Section 3, requiring coordination across software and configurable hardware, and controlled access in all parts of the system that configurability is allowed.

Reconfigurable architectures with process protection guarantees as well as existing reconfigurable architectures can be classified according to the customizability and the safety guarantee they provide. One possible range of configurable architecture classes spans a range of flexibility and safety as below:

1. **Full Configurability:** All processor components fully reconfigurable, all memory accesses checked and translated by a hardwired TLB which enforces OS mappings. All other elements of system configurable.
2. **Aggressive Configurability with safety:** All processor components excepting privilege mode changes and privileged operations fully reconfigurable, all memory accesses checked and translated by a hardwired TLB which enforces OS mappings. All other elements of system configurable, but accesses to registers, shared resources such as busses, and memories all controlled via hardwired access checking.
3. **Moderate Configurability with safety:** All processor components excepting privilege mode changes and privileged operations fully reconfigurable, all memory accesses checked and translated by a hardwired TLB which enforces OS mappings. Other devices which generate addresses are configurable, and have accesses checked by a shared (or multiple) TLB's. Configuration of accesses to registers, shared resources such as busses, and memories not allowed.
4. **Traditional Coprocessor Configurability:** Only coprocessor devices are configurable and their accesses to shared resources are unchecked (these could be checked by a hardwired TLB at the I/O interface). No address translation or shuffling in the MMU. This approach is typically taken for FPGA-based coprocessor configurable designs.
5. **Processor Configurability with safety:** All processor components excepting privilege mode changes and privileged operations fully reconfigurable, all memory accesses checked and translated by a hardwired TLB which enforces OS mappings. Other parts of the system are not configurable.

These architectures vary widely in their capabilities for customization to enhance application performance and the cost of providing multiprocess isolation guarantees. Because the issues are complex, and a detailed analysis of even one of these architectures is a topic for an entire technical paper, we merely point out that #1 allows the greatest flexibility, but cannot ensure that any isolation is guaranteed.

Architecture types #2 and #3 allow what one might consider to be a broad notion of useful configurability, leaving only the protection core, TLB checks, and a few key arbitration resources fixed. By maintaining minimal structures and mechanisms that have been identified as essential in satisfying process protection requirements, we believe that process protection can be guaranteed while allowing a certain degree of freedom in reconfigurability of the hardware. Within the scope of types #2 and #3 alone, there is a wide range of architectural space to explore.

Architecture type #4 is the traditional configurable coprocessor protection model where the configurable hardware is viewed as an extension of the system hardware, and dealt with by the operating system as an input/output device. This is dangerous, as the configurable logic can easily compromise system integrity. At a minimum, address checking (and interrupt capability) should be controlled.

Finally, architecture type #5 is the complement of #4, providing processor side configurability but no coprocessor

configurability. This type allows customization of data movement and computation around the primary locus of computation, and the tight coupling this makes customization significantly more powerful than in coprocessor systems. In type #5, process isolation is easily maintained by a hardwired TLB and checking all processor references.

5. Process Isolation in the Morph/AMRM Architecture

In the Morph/AMRM reconfigurable processor [4, 5], we propose that reconfigurable logic can be integrated into various components of the processor core to allow per applications adoption of optimal policies and/or custom mechanisms for data movement, memory hierarchy management, value prediction, branch prediction, etc. Rather than a more traditional approach of having a reconfigurable functional unit for custom computations, we are attaching reconfigurable logic to every component of the processor that is needed in optimizing various performance critical mechanisms and policies. This enables a highly flexible architecture but also makes virtually every part of the processor have reconfigurability.

The design of the Morph/AMRM protection architecture follows the protection model described in Section 3 by depending on memory addressing control and a privilege mode structure for ensuring that control and providing a virtual machine and system services for each process. However, because Morph/AMRM incorporates configurable logic deep internal to the processor core, careful engineering of exactly what must be hardwired is required. The Morph/AMRM architecture enables configurability in the processor complex with safe multiprocess protection. We also believe it enables much of the useful configurability in the processor complex, notably policies for improving efficient management of resources and even the addition of instructions, special functional units, or even processor states. The model provided to application programs is a private, configurable, virtual machine which enables rich application customization. These applications (and their customizations) are cleanly isolated.

The Morph/AMRM protection architecture is a type #3 configurable architecture (Moderate Configurability with Safety), preserving strong process isolation guarantees. The key idea is to have a few parts of the system be hardwired (unchangeable) and to also limit the connectivity to other resources (ensuring mediated access to those resources). Together, these two approaches ensure that key processor resources can be protected and recovered.

The basic model uses a context switching mechanism which synchronously switches processor state and all of the process' configurable hardware throughout the system simultaneously. Thus, Morph/AMRM eliminates concerns of software<->configurable hardware and configurable hardware<->configurable hardware interactions for unrelated programs. This leaves the main issues of ensuring secure context switching and strict address isolation.

The first two hardware features ensure virtualized execution and secure process switching. The latter five mechanisms enable process isolation.

1. Hardwired CP0 core:
The control processor is central in providing mechanisms such as privileged/user mode transitions and exception and interrupt delivering and handling for OS mediation, which are required to guarantee process isolation. The control processor core cannot be reconfigured in our design.

2. Hardwired instruction pipeline:
Controls and the structure of the execution pipeline are fixed for instruction sequencing. However, Customizable functional unit provided for custom instruction.

3. Hardwired CP0 to the TLB control for address translation and TLB entry management:
As pointed out in section 3, controlling/checking address translation in the TLB and TLB entry management is another central hardware requirement for process isolation. We have hardwired the TLB and the control from CP0 to TLB to guarantee correct isolation.

4. The remainder of the datapath, processor, and caches can all be configurable and connected in arbitrary ways for maximum flexibility.

5. TLB controlled accesses for other configurable elements accessing system bus:
Components such as system chip sets, I/O controllers, memory controllers that access the system bus(for memory or other memory-mapped items) can be fully configurable as long as they generate virtual addresses which is then checked by hardwired local TLBs.

6. Hardwired arbiters for controlling accesses to key shared resources:
Access to key shared interconnects such as the system bus are controlled by hardwired arbiters which are not changed, system reserves highest priority to allow preemption for these resources, configurable hardware can be redundant interconnects to these to accelerate, but cannot compromise the arbitration of these key resources.

7. Context switching and multiplexing/bypassing reconfigurable blocks to isolate and virtualize reconfigured logic blocks for different processes.

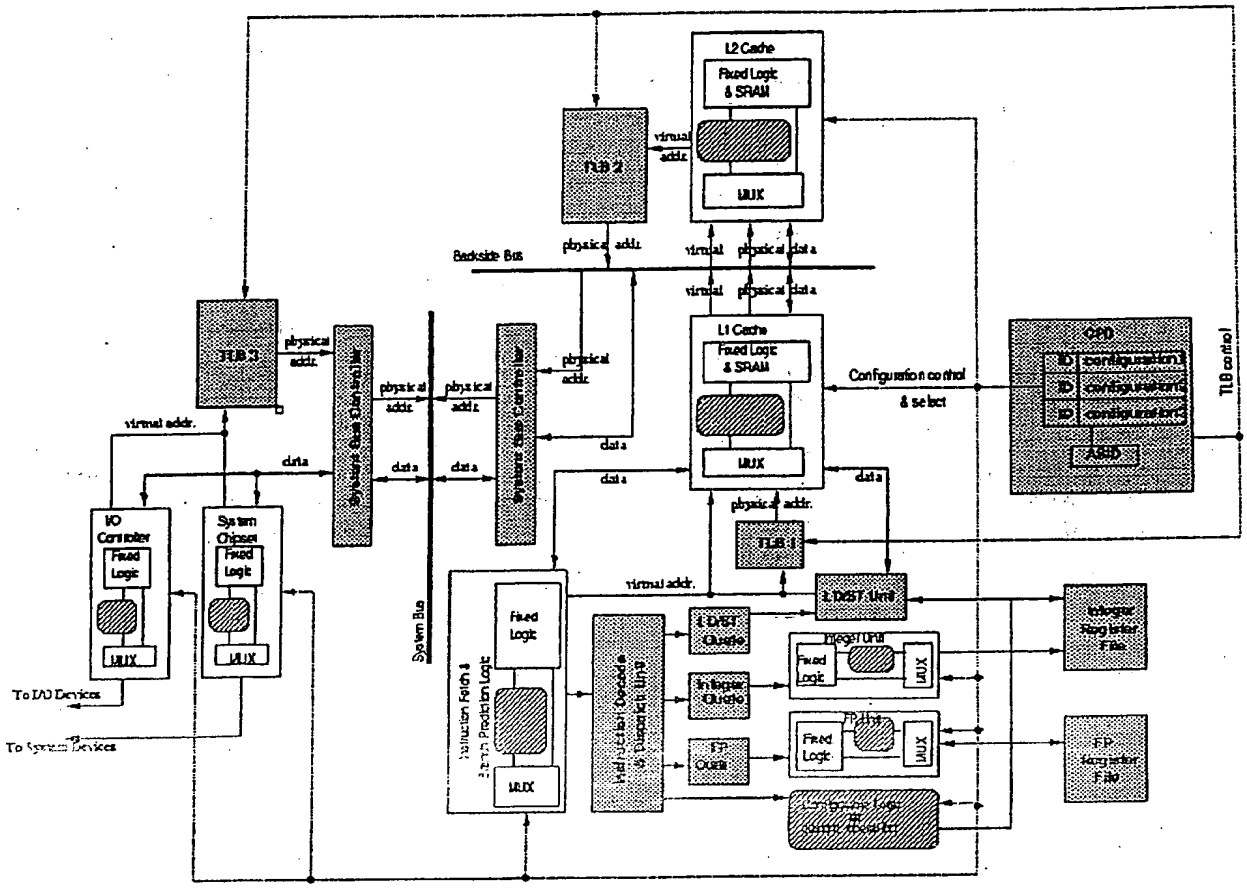

Figure 7. Morph/AMRM Protection Architecture. Rounded stripe box denotes reconfigurable logic block. Shaded box denotes fixed logic.

This is intentionally a simple model that provides most of the power of configurability and incurs a rather significant overhead of process isolation. The model provided to application programs is a private, configurable, virtual machine which enables rich application customization. These applications (and their customizations) are cleanly isolated. The schematic diagram of the protection architecture with the considerations listed above is given in Figure 7.

As explained earlier, these key features of the protection architecture are realized by restricting configurability of the components that are identified(in the previous sections) to be critical in maintaining classical process protection guarantees. The diagram in figure 7 shows which components are configurable or and which are not(shaded boxes in the diagram). System chip set, I/O controller, memory controllers also generate virtual addresses and have them checked and translated by a shared or local hardwired TLB, which also has hardwired control from CP0.

In addition, to eliminate newly introduced concerns of software<->configurable hardware and configurable hardware<->configurable hardware interactions, our basic model incorporates mechanisms to synchronously switch processor state and all of the process' configurable hardware throughout the system simultaneously. The outline of this mechanism, namely the synchronized hardware context switching, is briefly explained below:

Configuration owner table and configuration context register in CP0: The reconfigurable logic blocks are isolated, with multiplexers to control the inputs and outputs

to each block. Controls to these multiplexers come from CP0, which maintains a table of the owner processes of each reconfigured block. The configuration context register in CP0 indicates which reconfigurable blocks are to be used for the current process(see fig. 8). Notice that there are entries for two banks in each entry of the configuration owner table and that each reconfigurable logic block is divided into 2 banks. This can allow two different configurations of that block for two different processes to be switched without having to swap in the new configuration at each context switch.

Configuration selection/bypassing: If the entries in reconfigured block owner table for these blocks match the current process ID (different from process ID in OS. ASID used in TLB can be used here.), corresponding reconfigured block will be selected and activated while other reconfigurable blocks not used by the process will be bypassed. A more radical approach could even "suspend" the clock for this logic/memory to completely disable when the owner process is inactive it in order to ensure that no interference from unscheduled process' reconfigured logic.

Configuration swapping: If a block is to be used by the current process but does not match any of the two process ID fields (i.e. not configured for this process), an exception is raised and new configuration is swapped in.

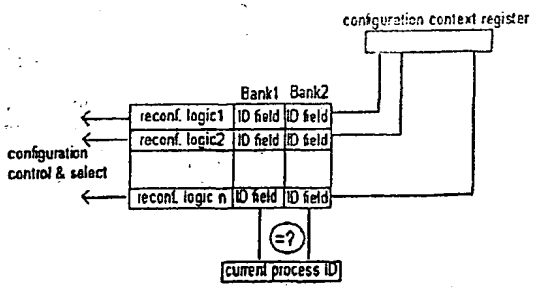

figure 8. Configuration owner table and configuration context register

6. Discussion and Related Work

The last decade has seen a proliferation of reconfigurable computing machines based on programmable logic blocks. In this section, we present some of these efforts and discuss the multiprocess protection issues in these alternate approaches in comparison with the Morph/AMRM architecture. We also discuss the limitations of our current Morph/AMRM protection architecture proposal.

FPGA processors, or processors built entirely out of FPGAs account for the majority of reconfigurable computing machines that have been proposed [11, 12, 13]. Like ASIC hardware, these processors perform special computations that are specific to the task that systems with these processors are to carry out. The difference between ASIC design and FPGA processors is that FPGA processors can have a few contexts so that these processors can carry out different operations in different stages of the task. Also, unlike the rigid ASIC designs, they can also re-tune themselves for better performance in response to the data that they are computing. Since most of these could not work as a stand-alone processor or only implemented as an experimental testbed, it is inappropriate to discuss multiprocess protection issues for these processors. There is no clear model for managing memory or external devices for these processors, which suggests that it will be difficult to, if not impossible, to design a stand-alone FPGA processor supporting safe multiprocess environment. Therefore, their use is usually limited to specific process engine used in domain-specific embedded systems.

While impressive performances have been reported for FPGA-based processors [11, 12, 13], these machines also have other shortcomings such as no instruction sequencing, long configuration time due to limit I/O, and slower implementation of standard functions. To overcome these shortcomings that make them less than ideal for general-purpose computing., architectures that couple a general-purpose control processor with FPGA co-processors have been proposed [14]. FPGA is placed as a slave computational unit on the same die as the processor and is used to speed up what it can, while the main processor controls the whole execution and takes care of other computations. Only some regular portions in the program such as loops and subroutines that can be programmed in reconfigurable logic to obtain speed-up are carried out in the reconfigurable part. This falls in the architecture type #4 as classified in section 4. The FPGA co-processor has its own memory interface and control logic, so it compromises multiprocess safety unless the system is extended so that the access is controlled by hardwired (local) TLB, which in turn is controlled through a hardwired control processor core. In Morph/AMRM, other configurable devices which generate addresses (e.g. system chip sets, input/output devices, memory controllers) must generate virtual address and is checked by local TLBs with fixed control from the control processor. Maintaining cache coherency is another problem to be solved for reconfigurable architectures of this kind.

Reconfigurable processors with dynamic instruction sets [15] try to extend the application-specific computation capability of a general processor with a computational unit implemented in reconfigurable logic. Again, these efforts have not explicitly addressed multiprocess protection. The overall architecture of the processor and the instruction execution cycle is similar to a general purpose processor but they have an extensible instruction set that can carry out

custom instructions as needed. As with other proposed works, this architecture is yet implemented only as an experimental testbed to demonstrate potential performance gain and thus lacks details in mechanisms to support real multiprocess environment. The existence of global controller in charge of interface to memory, registers, and processor status suggest that this architecture could be extended to provides protection guarantee of type #5 architecture. But the configurability is simply limited to providing configurable functional unit(implementing custom instructions) whereas in Morph/AMRM, configurability extends to other processor and system components to improve utilization of performance critical resources while providing each process with a private, configurable virtual machine by ensuring isolation and lock-up freedom. The reconfigurable logic blocks in the core enables customization of hardware granularity, memory system management, and bindings between resources, which is driven by the application.

The proposed Morph/AMRM protection architecture provides isolated, customizable virtual machine to each process, and pays a price in limiting configurability. While new instructions can be added, the parts of the instruction decoder and control that access the privilege control parts of the system must be hardwired. This still allows execution pipeline configurability and a wide range of optimizations, should they be performance sensible. The Morph/AMRM architecture also requires that all memory accesses be checked by hardwired TLBs and that there be no other address translation or shuffling beyond that. This restriction precludes adaptations that dynamically remap memory addresses at the translation level to implement scatter/gather technique and to increase the reach of TLBs [16]. However, such adaptations are not inherently safe, and depend on the values put into the translation tables. As such, they cannot be proven correct as a system attribute, but must depend on software to enforce some restrictions on use to ensure correctness and multiprocess isolation. Our Morph/AMRM architecture can be extended to include such a notion.

We have not completed the actual design and implementation of a prototype processor yet, but we are aware of the possibility that adding extra switches and muxes to isolate and context-switch customized logics that are spread about in the processor may incur considerable overhead in terms of clock rate, silicon area, and design complexity.

7. Summary and Future Work

Introducing application-controlled reconfigurability in the processor raises significant challenges in ensuring process isolation and protection (i.e. multiprocess isolations), a critical element of robust computing systems. In this paper, we have analyzed the implications of hardware reconfigurability on a multi-process environment and proposed architectural requirements for safe and protected execution for reconfigurable processors classified according to the protection guarantees and the level of reconfigurability they provide. Our study began by examining the protection structures of traditional processors and operating systems, identifying the key mechanisms of this protection architecture that is based on process isolation via address space isolation and mediation. This served as the starting point of our analysis and design of the protection architecture for reconfigurable processors.

Based on observations made through the analysis and classification, we have presented an architectural design incorporating a protection architecture that is best suited for our MORPH/AMRM reconfigurable processor. The key elements of this protection architecture were: 1. hardwired control processor for privilege mode transitions and instruction sequencing , 2. hardwired control to TLB for address translation and TLB entry management, 3. All other reconfigurable elements that generate addresses must deal in virtual address, and their accesses checked by local/shared TLBs, 4. Controlled access to key shared interconnects are controlled by hardwired arbiters. With these features, the OS and hardware mechanisms required for process process protection are well preserved and protected accesses are strictly checked and controlled by these mechanisms, while allowing all other components to be reconfigurable for better flexibility. This architectural design will thus provide most of the power of configurability and at the same time provide to each process a private, configurable, virtual machine which enables rich per-application basis adaptation. We plan to carry out simulations to verify our design and refine it in parallel with the development of our prototype evaluation board of the MORPH/AMRM processor. The simulations that we are planning on will be capable of revealing realistic OS – processor interaction, and is likely to be based on SimOS[17].

Making reconfigurable processors multiprocess-safe isn't the only requirement for a robust reconfigurable system, however. Dynamically validating and correcting the reconfigured logic is needed to find hardware faults and possibly to contain them. In the near future, we will study the issues concerning online validation/hardware fault containment and give a complete solution to building a reliable and robust reconfigurable system.

Acknowledgements

We would like to express our gratitude to Prof. Rajesh Gupta, Prof. Alex Nicolau, Prof. Alexander Veidenbaum, Louis Giannini, Ali Dasdan, Ben Zhang, and Martin Schulz for their comments and contributions. The Morph/AMRM project is supported by DARPA/ITO under contract number DABT63-98-C-0045 and by NSF Award number ASC-96-34947.

References

[1] Semiconductor Industry Association. National Technology Roadmap for Semiconductors(NTRS). 1997.

[2] Satapathy, R., Gupta, R. Analysis of Technology Trends: Making a Case for Architectural Adaptation in Custom Data-paths, 1997.

[3] A. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, X. Ji, Adapting Cache Line Size to Application Behavior. To appear in *Proceedings of the 13th ACM International Conference on Supercomputing, 1999(ICS '99)*.

[4] Chien, A. and Gupta, R. MORPH: A system Architecture for Robust High Performance Using Customization. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96)*(Oct. 1996), pp. 336-345.

[5] Zhang, X., Dasdan, A., Schulz, M., Gupta, R., and Chien, A. Architectural Adaptation for Application-Specific Locality Optimization. In *Proceedings of the International Conference on Computer Design* (Oct. 1997)

[6] DeHon, A. *Reconfigurable Architectures for General-Purpose Computing*, Ph.D. thesis, Massachusetts Institute of Technology, 1996

[7] Kiczles, G., et Al. Open Implementation Design Guidelines. *In Proceedings of the 19th International Conference on Software Engineering*, 1997.

[8] MIPS technologies, Inc. *MIPS R10000 Microprocessor User's Manual*, 1996.

[9] Bach, M., *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[10] Leffler, S., McKusick, M., Karels, M., Quarterman, J. *The Design and Implementation of the 4.3BSD UNIX Operating System, Addison-Wesley, Reading, MA, 1989.*

[11] Gokhale, M., Holmes, W., Kosper, A., Lucas, S., Minnich, R., Sweely, D., and Lopresti, D. Building and Using a Highly Programmable Logic Array. *IEEE Computer*, 24(1):pp. 81-89, Jan. 1991.

[12] Arnold, J., Buell, D., and Davis, E., Splash 2. In Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 316-324, June 1992.

[13] Vuillemin, J., Bertin, P., Roncin, D., Shand, M., Touati, H., and Boucard, P. Programmable Active Memories: Reconfigurable Systems Come of Age. *IEEE Transactions on VLSI Systems*, 4(1):pp.56-69, Mar. 1996.

[14] Hauser, J. and Wawrzynek, J. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1997.

[15] Wirthlin, J. and Hutchings, B. DISC: The dynamic instruction set computer. In *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*, John Schewel, Editor, Proc. SPIE 2607, pp. 92-103 (1995).

[16]. J. Carter, W. Hsieh, L. Stoller, M. Swanson, L. Zhang, E. Brunvand, A. Davis, Chen-Chi Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. Impulse: Building a Smarter Memory Controller. To appear in the *Proceedings of IEEE Fifth International Symposium on High Performance Computer Architecture (HPCA-5)*

[17] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta. Complete Computer Simulation: The SimOS Approach. In *IEEE Parallel and Distributed Technology*, Fall 1995.